

```

%% Zuerst die Funktion zur Berechnung des 1D-Minimum mit goldenem Schnitts
function res = goldenerschnitt(f,a,b,TOL)
% f: R --> R
% a, b Intervallgrenzen (a < b), TOL Toleranz fuer minimale Intervalllaenge
%
% berechnet Minimum von f innerhalb [a,b] nach dem Verfahren des goldenen
% Schnitts

```

```

% BEACHTE: In jedem Schritt nur eine neue Funktionsauswertung.

```

```

s1 = (3-sqrt(5))/2;
s2 = 1-s1; % = (sqrt(5)-1)/2;

```

```

v = a + s1*(b-a);
w = a + s2*(b-a);
fv = f(v);
fw = f(w);

```

```

while (abs(b-a) >= TOL)

```

```

    if (fv < fw)
        b = w;
        w = v; fw = fv;
        v = a + s1*(b-a);
        fv = f(v);
    else
        a = v;
        v = w; fv = fw;
        w = a + s2*(b-a);
        fw = f(w);
    end

```

```

end

```

```

end

```

```

if(fw < fv)
    res = w;
else
    res = v;
end

```

```

end

```

```

%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%

```

```

%% Dann eine Funktion, die fuer gegebene Funktion den Gradienten ausrechnet

```

```

function res = grad(f,x)

```

```

% Gradient der Funktion f: R^n --> n an der Stelle x mit zentralem Differenzenquotient (s. NumerikI)

```

```

n = length(x);
res = zeros(n,1);
h = 1e-8;%bei Differenzenquotient waehle h=sqrt(Maschinengenauigkeit) (s. NumerikI)
for j=1:n
    x(j)=x(j)+h;
    tmp=f(x);
    x(j)=x(j)-2*h;
    res(j) = 1/(2*h)*(tmp-f(x));
    x(j)=x(j)+h;
end

```

```

%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%

```

```

%% und natuerlich die zu minimierende Funktion selbst

```

```

function res = funPP3a(x)

```

```

% unsere Beispielfunktion

```

```

x1 = x(1);
x2 = x(2);
res = 100.*(x2-x1.^2).^2+(1-x1).^2;

```

```

end

```

```

%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%

```

```

%% PP3a)

```

```
function x_k = steilstersabstieg(f,x0,k,TOL)
% f:      R^n --> R
% x0      nx1-Startvektor
% k       >= 1, Anzahl Iterationen
% TOL     Toleranz fuer eingebettete 1-dim. Minimierung
% GOLD    Boolean; gibt an, ob goldener Schnitt oder quadrat. Interpolation zur 1-dim. Minimierung benutzt
%         wird
%
% Verfahren des steilstens Abstiegs
```

```
format long
```

```
    n          = length(x0);
    x_k         = x0;
    xvis        = zeros(n,k+1);
    xvis(:,1)   = x_k;
```

```
for j = 1:k
```

```
    d_k        = -grad(f,x_k);      % nx1
    %d_k        = -[-400*(x_k(2)-x_k(1)^2)*x_k(1) - 2*(1-x_k(1)); 200*(x_k(2)-x_k(1)^2)];
    f_temp      = @(a) f(x_k+a*d_k); % R --> R
    a_k         = goldenerschnitt(f_temp,0,1e-2,TOL); % skalar

    x_k         = x_k + a_k*d_k;

    xvis(:,j+1) = x_k;
```

```
end
```

```
% Visualisierung des vorliegenden Beispiels (fuer n = 2 und konkretes f)
hold on;
[X,Y] = meshgrid(-1.5:.05:2,-.5:.05:2);
Z = 100.*(Y-X.^2).^2+(1-X).^2;
contour(X,Y,Z,[-1:.2:3].^2);
plot(xvis(1,:),xvis(2:,:),'-.')
hold off;
```

```
end
```

```
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% PP3b)
```

```
function x_k = steilstersabstieg_LGS(A,b,x0,k)
```

```
% A   nxn-Matrix, spd
% b   nx1-Vektor
% x0  nx1-Startvektor
% k   Anzahl der Iterationsschritte
%
```

```
% Verfahren des steilstens Abstiegs zur Loesung von Ax = b;
% alternativ benutze die zug. Linearform (funLGS.m) oder das beigegefuegte cg-Verfahren cg.m
```

```
format long
```

```
    n          = length(x0);
    x_k         = x0;
    xvis        = zeros(n,k+1);
    xvis(:,1)   = x_k;
```

```
for j = 1:k
```

```
    d_k        = b - A*x_k;          % einfache Darstellung des Gradienten im Falle eines LGS
    a_k        = d_k'*d_k / (d_k'*A*d_k); % alpha_k nach Skript
    x_k        = x_k + a_k*d_k;

    xvis(:,j+1) = x_k;
```

```
end
```

```
% Visualisierung fuer das vorliegende Beispiel
hold on;
[x1,x2] = meshgrid(-20:1:20,-20:1:20);
contour(x1,x2,x1.^2 + 20*x2.^2,[0:5:400].^2);
plot(xvis(1,:),xvis(2:,:),'-*','Linewidth',2)
```

```

hold off;

end
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%% cg-Verfahren
function [x_k j] = cg(A,b,x0,TOL,k)
%in Arbeit
end
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%% Aufruf von PP3a und PP3b sowie zum Vergleich der erste Teil von PP3c
clear all
close all

x0_10a    = [-1.2 1]';

% PP3 a), wobei 1-dimensionale Minimierung mit goldenem Schnitt und Toleranz 1e-12
% 50 Iterationen
figure(1)
title('PA10 (b) mit goldenem Schnitt - k = 50')
x_PA10b_gold = steilstersabstieg(@funPP3a,x0_10a,50,1e-12)

A        = [1 0; 0 20];
b        = [0 0]';
x0_LGS   = [20 1]';

% LGS mit Methode des steilsten Abstiegs
% 50 Iterationen
figure(2)
title('LGS - Methode des steilsten Abstiegs - k = 50')
x_LGS_MdsA = steilstersabstieg_LGS(A,b,x0_LGS,50)

% LGS mit cg-Verfahren = PP3c) Teil 1
% 2 Iterationen visualisiert, gerechnet mit Toleranz 1e-12
% (gemaess (5.1) liefert cg-Verfahren fuer vorliegendes A nach 2 Schritten
% bereits exakten Wert)
figure(3)
title('LGS - cg-Verfahren - k = 2')
[x_LGS_cg j]= cg(A,b,x0_LGS,1e-12,2)
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%% Nochmal cg, modifiziert auf Teil 2 von Aufgabe PP3 c)
function [x_k j]= cg2(A,b,x0,TOL)
%in Arbeit
end
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%% Unvollstaendige Cholesky-Zerlegung der Matrix A
function L = uChol(A)

n = length(A);
L = zeros(n,n);

for k = 1:n
    L(k,k) = sqrt(A(k,k) + sum(L(k,1:k-1).^2));
    for i=k+1:n
        if (A(i,k) == 0)
            L(i,k) = 0;
        else
            L(i,k) = (A(i,k) - sum(L(i,1:k-1).*L(k,1:k-1)))/L(k,k);
        end
    end
end
end
end
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%% PP2d
function [xk k] = cg_vorkond(A,b,x0,TOL)
    L=uChol(A);

```

```

xk = x0;
gk = A*x0-b;
temp = L'\gk;
dk = L\temp;
%dk = B*gk;
% wobei B = inv(L)'\*inv(L);

normb = norm(b);
TOLnormb = TOL*normb;
k=1;

rhok = gk'*dk;

while (norm(rhok)>TOLnormb)
    k=k+1;
    Adk = A*dk;
    Adkdk = Adk'*dk;
    alphak = rhok/Adkdk;

    xk = xk - alphak*dk;
    gk = gk - alphak*Adk;

    temp = L'\gk;
    Bgk = L\temp;
    %Bgk = B*gk;

    rhok_temp = rhok;
    rhok = gk'*Bgk;

    betak = -rhok/rhok_temp;
    dk = Bgk - betak*dk;

end
end
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Ausgabe PP3 c) Teil 2 und PP3 d)
clear all
close all

A1 = MatrixGenerator(4); A2 = MatrixGenerator(20); A3 = MatrixGenerator(40);
b1 = (1:16)'; b2 = (1:400)'; b3 = (1:1600)';
x01 = ones(16,1); x02 = ones(400,1); x03 = ones(1600,1);

[xk1 j1] = cg2(A1,b1,x01,1e-12);
[xk2 j2] = cg2(A2,b2,x02,1e-12);
[xk3 j3] = cg2(A3,b3,x03,1e-12);
[xk1 k1] = cg_vorkond(A1,b1,x01,TOL,L1);
[xk2 k2] = cg_vorkond(A2,b2,x02,TOL,L2);
[xk3 k3] = cg_vorkond(A3,b3,x03,TOL,L3);

x1 = A1\b1;
x2 = A2\b2;
x3 = A3\b3;

norm(xk1-x1)
norm(xk2-x2)
norm(xk3-x3)
j1
j2
j3

norm(xk1-x1)
norm(xk2-x2)
norm(xk3-x3)

k1
k2
k3
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FOM
function R = fom(A,b,TOL)

```

```

n = length(A);
normb = norm(b);
V(:,1) = b / normb;
R(1) = normb;
k = 1;
normrk = normb;

while (normrk > TOL)
    % Arnoldi-Schritt
    vtilde = A*V(:,k);
    for j=1:k
        H(j,k) = V(:,j)'\*vtilde;
        vtilde = vtilde - H(j,k)*V(:,j);
    end
    H(k+1,k) = norm(vtilde);
    if(H(k+1,k) <= TOL)
        break
    end
    V(:,k+1) = vtilde / H(k+1,k);

    % FOM
    e1 = eye(k,1);
    y = H(1:k,:)\(normb*e1);
    x = V(:,1:k)*y;
    normrk = norm(b - A*x);
    R(k+1) = normrk;
    k = k+1;
end

```

```

end
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%% GMRES
function R = gmres(A,b,TOL)

```

```

n = length(A);
normb = norm(b);
V(:,1) = b / normb;
R(1) = normb;
k = 1;
normrk = normb;

while (normrk > TOL)
    % Arnoldi
    vtilde = A*V(:,k);
    for j=1:k
        H(j,k) = V(:,j)'\*vtilde;
        vtilde = vtilde - H(j,k)*V(:,j);
    end
    H(k+1,k) = norm(vtilde);
    if(H(k+1,k) <= TOL)
        break
    end
    V(:,k+1) = vtilde / H(k+1,k);

    % GMRES
    e1 = eye(k+1,1);
    [Qk Rk] = qr(H);
    reSe = normb*Qk'\*e1;
    reSe = reSe(1:k);
    y = Rk(1:k,:)\reSe;
    x = V(:,1:k)*y;
    normrk = norm(b - A*x);
    R(k+1) = normrk;
    k = k+1;
end

```

```

end
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%% vorkonditioniertes GMRES
function R = gmres_vorkond(A,b,TOL)

```

```

%[L U] = unvLU(A,1e-6);
[L U P] = ilu(A,struct('type','ilutp','droptol',1e-6));
%M = inv(L*U);

temp = L\(P*b);
Mb = U\temp;
% Mb = M*b;

normMb = norm(Mb);
V(:,1) = Mb / normMb;
R(1) = normMb;
k = 1;
normrk = normMb;

while (normrk > TOL)
    Avk = A*V(:,k);
    temp = L\(P*Avk);
    vtilde = U\temp;
    %vtilde = M*A*V(:,k);
    for j=1:k
        H(j,k) = V(:,j)'*vtilde;
        vtilde = vtilde - H(j,k)*V(:,j);
    end
    H(k+1,k) = norm(vtilde);
    if(H(k+1,k) <= TOL)
        break
    end
    V(:,k+1) = vtilde / H(k+1,k);

    % GMRES
    e1 = eye(k+1,1);
    [Qk Rk] = qr(H);
    reSe = normMb*Qk'*e1;
    reSe = reSe(1:k);
    y = Rk(1:k,:)\reSe;
    x = V(:,1:k)*y;

    normrk = norm(b - A*x);
    R(k+1) = normrk;
    k = k+1;
end

end
%
%% Ausgabe fuer PP3 e)
clear all
close all

load west0479
A = west0479;
[m n] = size(A);
figure(1)
spy(A)

b = sum(A,2);
normb = norm(b);
%x = A\b

figure(2)
subplot(1,2,1)
RFOM = fom(A,b,1e-8);
plot(1:length(RFOM),RFOM)
title('FOM')
subplot(1,2,2)
RGMRES = gmres(A,b,1e-8);
plot(1:length(RGMRES), RGMRES)
title('GMRES ohne Vorkonditionierung')

figure(3)
RGMRESkond = gmres_vorkond(A,b,1e-8);
plot(1:length(RGMRESkond), RGMRESkond)
title('GMRES mit unvollst. LU-Zerlegung als Vorkonditionierung')

```

```
% RFOI(470:479)
% RGMRES(470:479)
RGMRESkond
```

```
%Veranschaulichung der Struktur von L und U
figure(4)
%[L U] = unvLU(A,1e-6);
[L,U,P] = ilu(A,struct('type','ilutp','droptol',1e-6));
subplot(1,2,1)
spy(L)
subplot(1,2,2)
spy(U)
```